Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3210

Цыпандин Н. П.

Преподаватели:

Косяков М.С.

Тараканов Д. С.

Санкт-Петербург

2022

Задача №1 «Агроном-любитель»

```cpp
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n;
    int left = 1, right, ma = 0;
    int flower, current = -1, count = 1, start_pos = 1;
    for (int i = 1; i <= n; ++i) {
        cin >> flower;
        if (flower == current) {
            count++;
        } else {
            current = flower;
            count = 1;
        }
        if (count == 3) {
            if ((i - start_pos) > ma) {
                ma = i - start_pos;
                left = start_pos;
                right = i - 1;
            }
            start_pos = i - 1;
            current = flower;
            count = 2;
        }
    }
    if ((n - start_pos + 1) > ma) {
        left = start_pos;
        right = n;
    }
    cout << left << ' ' << right << endl;
    return 0;
}
```

Пояснение к примененному алгоритму:

Храним счетчик на подряд повторяющиеся элементы, и его значение (номер цвета). Также ответ left и right и максимальное значение длины для обновления ответа. Проходя по массиву данных, поддерживаем эти значения для решения задачи. Если оказалось, что у нас набралось 3 подряд одинаковых цвета, значит это новый претендент на ответ, проверяем получившуюся длину, если она больше нашего текущего максимума, обновляем значения. Далее продолжим считать сдвинув левый указатель

Асимптотика: O(N)

Задача №2 «Зоопарк Глеба»

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

bool is_animal(char c);

bool is_trap(char c);

bool is_animal_and_trap(char a, char b);

bool equals_ignore_case(char a, char b);

struct zoo_pointer {
    char c;
    size_t index;
};

int main() {
    string zoo;
    cin >> zoo;

    size_t len = zoo.length();
    bool ans = true;

    /**
     * Если это палиндром относительно прописных и строчных букв, то выводим ответ
     * Иначе проверим на правильную скобочную последовательность
     */
    for (size_t i = 0; i < len / 2; ++i) {
        if (!is_animal_and_trap(zoo[i], zoo[len - i - 1]) || !equals_ignore_case(zoo[i],
zoo[len - i - 1])) {
            ans = false;
            break;
        }
    }
    if (ans) {
        cout << "Possible" << endl;
        for (size_t i = len / 2; i >= 1; --i) {
            cout << i << ' ';
        }
        return 0;
    }

    /**
     * Если строка - правильная скобочная последовательность относительно строчных и прописных букв,
     * тогда мы можем отобразить это на окружность и провести непересекающиеся линии
     * a, b, ..., z <=> "("
     * A, B, ..., Z <=> ")"
     */

    vector<struct zoo_pointer> balanced_entity_stack;
    vector<size_t> indexes(len / 2);
    size_t trap_id = 0, animal_id = 0;

    for (size_t i = 0; i < len; ++i) {
        trap_id += is_trap(zoo[i]);
        animal_id += is_animal(zoo[i]);

        size_t assigned_index = trap_id;
        if (is_animal(zoo[i]))
            assigned_index = animal_id;

        if (balanced_entity_stack.empty()) {
            balanced_entity_stack.push_back({.c = zoo[i], .index = assigned_index});
            continue;
        }
```

```
        struct zoo_pointer current = balanced_entity_stack[balanced_entity_stack.size() - 1];
        char top = current.c;

        if (is_animal_and_trap(top, zoo[i]) && equals_ignore_case(top, zoo[i])) {
            if (is_trap(zoo[i]))
                indexes[trap_id - 1] = current.index;
            else
                indexes[current.index - 1] = animal_id;
            balanced_entity_stack.pop_back();
            continue;
        }
        balanced_entity_stack.push_back({.c = zoo[i], .index = assigned_index});
    }

    if (!balanced_entity_stack.empty()) {
        cout << "Impossible" << endl;
    } else {
        cout << "Possible" << endl;
        for (size_t index: indexes)
            cout << index << ' ';
    }
    return 0;
}

bool is_animal(char c) {
    return c >= 'A' && c <= 'Z';
}

bool is_trap(char c) {
    return c >= 'a' && c <= 'z';
}

bool is_animal_and_trap(char a, char b) {
    return (is_trap(a) && is_animal(b)) || (is_trap(b) && is_animal(a));
}

bool equals_ignore_case(char a, char b) {
    if (is_trap(a))
        a = char(a - 'a');
    else
        a = char(a - 'A');
    if (is_trap(b))
        b = char(b - 'a');
    else
        b = char(b - 'A');
    return a == b;
}
```

Пояснение к примененному алгоритму:

Заметим, что если введенная строка палиндром, то мы можем отобразить животных и ловушки на окружности и провести непересекающиеся линии.

Иначе, проверим строку на правильную скобочную последовательность, где прописная буква это открывающаяся скобка, а соответствующая строчная буква это закрывающаяся скобка. Тогда мы тоже можем отобразить все это дело на окружности. Для этого заведем импровизированный стэк и вспомогательный массив для вывода индексов. Если в конце окажется что стэк balanced_entity_stack пустой, значит невозможно, иначе выводим ответ.

Асимптотика: O(N)

Задача №3 «Конфигурационный файл»

```
#include <iostream>
#include <string>
#include <map>
#include <algorithm>
#include <vector>


using namespace std;

bool block_start(const string &line);

bool block_end(const string &line);

bool number_assign(string line);

bool temp_assign(string line);

int parse_number(string line);

string parse_temp(string line);

string parse_second_temp(string line);

int main() {
    map<string, vector<int>> state_holder;
    vector<string> assigned_temps;
    vector<int> assigned_temps_cnt = {0};
    string line;
    while (cin >> line) {
        if (block_start(line)) {
            assigned_temps_cnt.push_back(0);
            continue;
        } else if (block_end(line)) {
            int cnt = assigned_temps_cnt.back();
            for (size_t i = assigned_temps.size() - 1; cnt > 0; --i) {
                state_holder.find(assigned_temps[i])->second.pop_back();
                assigned_temps.pop_back();
                --cnt;
            }
            assigned_temps_cnt.pop_back();
        } else if (number_assign(line)) {
            string temp1 = parse_temp(line);
            int number = parse_number(line);
            if (state_holder.find(temp1) == state_holder.end()) {
                state_holder[temp1].push_back(0);
            }
            assigned_temps_cnt.back()++;
            assigned_temps.push_back(temp1);
            state_holder[temp1].push_back(number);
        } else if (temp_assign(line)) {
            string temp1 = parse_temp(line);
            string temp2 = parse_second_temp(line);
            if (state_holder.find(temp1) == state_holder.end()) {
                state_holder[temp1].push_back(0);
            }
            if (state_holder.find(temp2) == state_holder.end()) {
                state_holder[temp2].push_back(0);
            }
            assigned_temps_cnt.back()++;
            assigned_temps.push_back(temp1);
            int value = state_holder[temp2].back();
            state_holder[temp1].push_back(value);
            cout << value << endl;
        }
    }
```

```
    return 0;
}

bool block_start(const string &line) {
    return line == "{";
}

bool block_end(const string &line) {
    return line == "}";
}

bool number_assign(string line) {
    return line[line.length() - 1] >= '0' && line[line.length() - 1] <= '9';
}

bool temp_assign(string line) {
    return !number_assign(std::move(line));
}

int parse_number(string line) {
    size_t i = line.length() - 1;
    int digit = 1, res = 0;
    while (line[i] != '=') {
        if (line[i] == '-') {
            res *= -1;
            break;
        }
        res = res + digit * (line[i] - '0');
        digit *= 10;
        --i;
    }
    return res;
}

string parse_temp(string line) {
    string res;
    size_t i = 0;
    while (line[i] != '=') {
        res.push_back(line[i]);
        ++i;
    }
    return res;
}

string parse_second_temp(string line) {
    string res;
    size_t i = line.length() - 1;
    while (line[i] != '=') {
        res.push_back(line[i]);
        --i;
    }
    reverse(res.begin(), res.end());
    return res;
}
```

Пояснение к примененному алгоритму:

В мапе будем хранить присвоенные значения для каждой из переменных. Также будем поддерживать stack присвоенных переменных вообще и массив для подсчета количества присвоенных переменных внутри одного scope’а для того, чтобы в конце scope’а вернуть старые значения переменным. Для вывода значения ходим в мапу.

Асимптотика зависит от запросов:

Для присваиваний log(N)

Для выхода из scope N*log(N)

Задача №4 «Профессор Хаос»

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    long long A, B, C, D, k;
    cin >> A >> B >> C >> D >> k;

    long long n = A, cnt = 0;
    vector<long long> cycle = {A};

    while (cnt < k) {
        n = n * B;

        if (n <= C) {
            cout << 0 << endl;
            return 0;
        }

        n = n - C;

        if (n > D) {
            cout << D << endl;
            return 0;
        }

        if (n == A) {
            cout << cycle[k % (cnt + 1)] << endl;
            return 0;
        }

        ++cnt;
        cycle.push_back(n);
    }
    cout << n << endl;
    return 0;
}
```

Пояснение к примененному алгоритму:

Заметим, что a <= d, так же заметим, что одна итерация (функция) монотонна (в зависимрсти от B и C). Если мы после вычитания C получили число большее D, очевидно, что в следующей итерации мы получим такой же результат.

Так же область допустимых значений ограничен D, тогда мы можешь хранить в массиве один период, и если мы получили снова A, то обратимся к массиву что бы получить ответ.

Асимптотика: O(D)